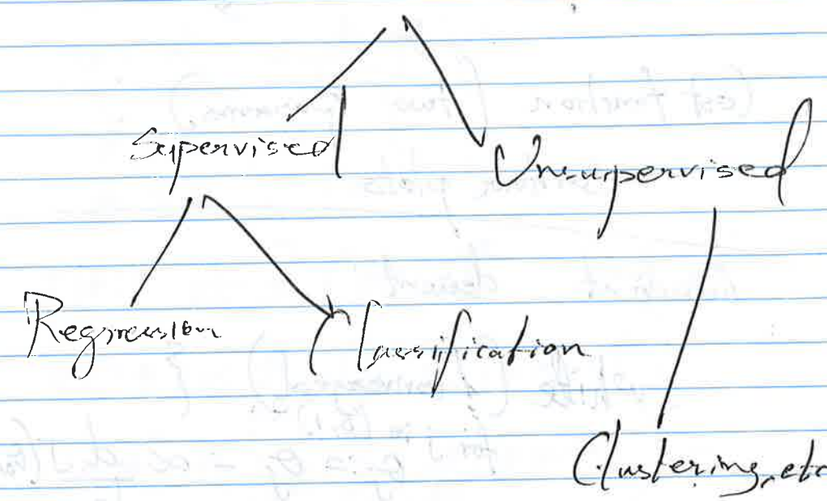
ML Coursera:

Week 1:

Linear regression



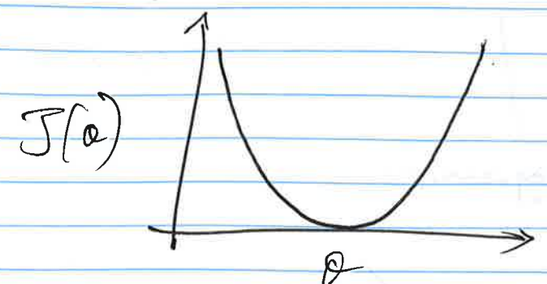
Linear regression (one variable):

$h(x) = \theta_0 + \theta_1 x$

Cost function:

$$\underset{\theta_0, \theta_1}{\text{Minimize}} \; J(\theta_0, \theta_1) = \frac{\sum_{i=1}^{m} \left(h(x_i) - y_i\right)^2}{2m}$$

Cost function (one param):



Cost function (two params):
- Contour plots

Gradient descent:

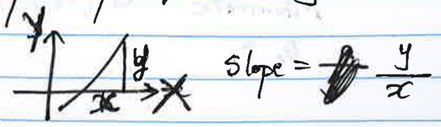
while (! converged) {
  for j in (0,1):
    $\theta_j := \theta_j - \alpha \frac{d}{d\theta_j} J(\theta_0, \theta_1)$
}

α - learning rate

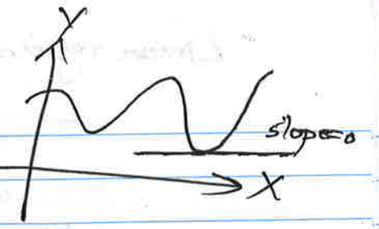
→ Simultaneous update

$\frac{d}{d\theta}(J(\theta))$ → Change in cost fn. for a small change in θ

↘ Slope of the tangent



At local minimum:
(one param)



$\theta := \theta - \alpha \frac{d}{d\theta}(J(\theta))$

Say $J(\theta) = \theta x$

At local minimum, $x = 0$.

$\theta := \theta - \alpha \frac{d}{d\theta}(\theta x)$

$\theta := \theta - \alpha x$

$\theta := \theta$ $(x = 0)$

$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (\theta x - y)^2$ (here, $h_\theta(x) = \theta x$)

$\theta := \theta - \alpha \frac{d}{d\theta} \left(\frac{1}{2m} \sum_{i=1}^{m} (\theta x - y)^2 \right)$

$\theta := \theta - \alpha \frac{1}{2m} \sum_{i=1}^{m} \left(\frac{d}{d\theta} (\theta x - y) \right)$

$\theta := \theta - \alpha \frac{1}{m} \sum_{i=1}^{m} (x)$

$x = 0$ at local minimum

$\theta := \theta$

$\theta := \theta - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (\theta x - y) \cdot x$ (Chain rule)

At local minima, θx is as close to y as possible.
Ideal case, $\theta x = y$.

Linear regression cost function:

— Convex function

↓ ~~Only one local.~~

No local minima, only global minimum

~~Batch Gradient~~

Batch gradient descent

↳ Look at all training examples during each iteration

$\left(\sum_{i=1}^{m} (h_\theta(x_i) - y_i)\right)$

Other scenario — look at subset of training data in each iter

---

Gradient descent for linear regression:

$h_\theta(x) = \theta_0 + \theta_1 x$ (Hypothesis / model)

$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$ (Cost function)

$\theta_0 := \theta_0 - \alpha \frac{d}{d\theta_0} \left[\frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2\right]$

$\theta_0 := \theta_0 - \alpha \cdot \frac{1}{m} \cdot \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$ (Chain rule)

$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} \left[\frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2\right]$

$\theta_1 := \theta_1 - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) \cdot x_i$ (Chain rule)

WEEK 2:

Multivariate Linear Regression:

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 + \ldots + \theta_n x_n$

$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$ (Here, $x_0 = 1$)

(Feature vector)

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$

(Parameter vector)

$h_\theta(x) = \theta^T x$

Cost fn.: $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2$

Gradient descent:

Repeat {

$\theta_j := \theta_j - \alpha \frac{d}{d\theta_j} J(\theta)$

(Simultaneously update all θ_j for $j = 0, 1, \ldots n$)

}

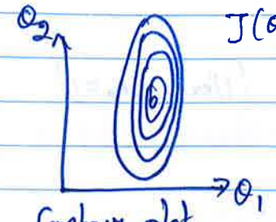
$\theta_j = \theta_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) \cdot x_j^i$

Feature scaling:

↳ Scale features to make gradient descent converge faster

(eg) x_1 = size (0–2000 $feet^2$)

x_2 = number of bedrooms (1–5)



Contour plot

↓

This makes convergence slow

Scale features: $x_1 = \frac{\text{size}(feet^2)}{2000}$

$x_2 = \frac{\text{no. of bedrooms}}{5}$

$0 \le x_1 \le 1$; $0 \le x_2 \le 1$

→ Converges faster

Mean normalization:

Replace x_i with $x_i - \mu_i$

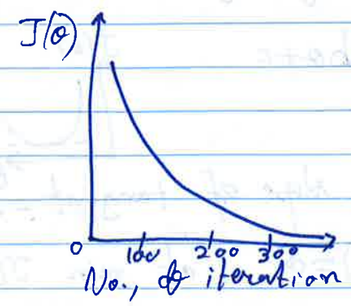
(μ_i → mean of the feature value over all training examples)

In general, $x_i \leftarrow \frac{x_i - \mu_i}{s_i}$

(s_i → Range or std deviation)

Range will have the impact of outliers, std deviation handles outliers better

Gradient descent – Learning rate:



(eg) Declare convergence if $J(\theta)$ decreases by less than ϵ (small value) in one iteration

If $J(\theta)$ is increasing, choose smaller value for α (Learning rate)

→ α too small: slow convergence

→ α too large: may not converge.

Features and polynomial regression:

~~Quadratic model~~

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x^2 + \theta_3 x^3$

(Cubic model)

↓

Make this cubic function a linear model by choosing features as

Features $\begin{cases} x_1 = x \\ x_2 = x^2 \\ x_3 = x^3 \end{cases}$ → Parts of cubic fn.

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$

↳ Now apply linear regression

(Feature scaling is extremely important in this case)

Normal equation:

Solve for θ analytically w/o iterations

$J(\theta) = a\theta^2 + b\theta + c$

(cost fn.

At min $J(\theta)$, slope of tangent $= 0$

$\Rightarrow \frac{\partial}{\partial \theta} J(\theta) = 0$ at min $J(\theta)$

Solve for θ.

This is for $1 \Rightarrow \theta$.

For $\theta \in R^{n+1}$

$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2$

Set: $\frac{\partial}{\partial \theta_j} J(\theta) = 0$ (for every j)

Solve for $\theta_0, \theta_1, \ldots, \theta_n$

$X \rightarrow$ Matrix of all training data

$X = \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}$ } training examples

features

$y = \begin{bmatrix} \\ \\ \end{bmatrix}$

↳ m dimensional vector

$\theta = (X^T X)^{-1} X^T y$

$X\theta = Y$

For X to have inverse, X has to be a square matrix.

So $\theta = X^{-1}Y$ may not always be possible.

$\therefore X^T X \theta = X^T Y$

$X^T X$ is a square matrix

$\Rightarrow \boxed{\theta = (X^T X)^{-1} X^T Y.}$

Normal equation

Feature scaling isn't necessary for normal equation method (since no iterations, or gradient descent, or learning rate involved).

With gradient descent, w/o feature scaling a value of α (learning rate) that is applicable for one feature, may not apply for other features.

Gradient Descent	Normal equation
→ Need to choose α	→ No α
→ Needs many iteration	→ No iterations
→ Works well when no. of features (n) is large.	→ Expensive if n is large. $(X^T X)^{-1}$ → $n \times n$ matrix ↳ Large time complexity

What if X^TX is non-invertible?

Could be due to:

↳ Redundant (linearly dependant) features

↳ Too many features ($m \leq n$)

Delete some features

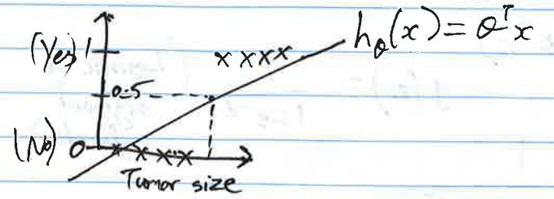
Regularization

Vectorization:

Gradient descent:

Week 3:

Classification:

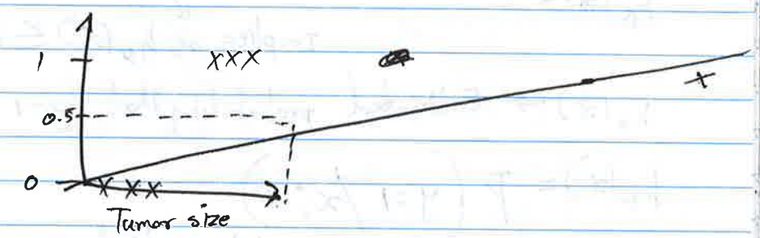


Apply Linear regression → threshold classifer

If $h_\theta(x) >= 0.5$, predict "$y = 1$"

If $h_\theta(x) < 0.5$, predict "$y = 0$"

But, outliers change the regression model:



One outlier example can have a big change in model.

Also, linear regression can result in

$h_\theta(x) > 1$ or $h_\theta(x) < 0$,

but in classification, $y = 0$ or 1.

⇒ Linear regression isn't applicable for classification problems.

Logistic regression : → Want $0 \le h_\theta(x) \le 1$

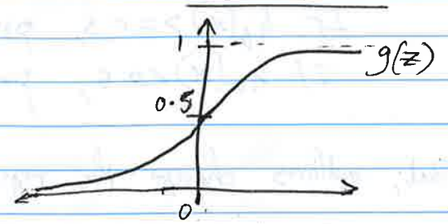
~~$h_\theta(x) = g(\theta x)$~~

$h_\theta(x) = g(\theta^T x)$

$g(z) = \frac{1}{1+e^{-z}}$ (Logistic or sigmoid function)

$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

Sigmoid fn. :



~~$h_\theta(x)$~~

↓ Implies $0 \le h_\theta(x) \le 1$

$h_\theta(x)$ → Estimated probability that $y=1$ for x.

$h_\theta(x) = P(y=1 / x; \theta)$

(probability that $y=1$ given x, parametrized by θ)

~~$P(y)$~~ $P(y=0 / x; \theta) = 1 - P(y=1 / x; \theta)$

When $z=0$, $g(z) = 0.5$

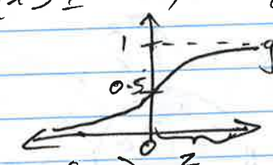
~~$z \to \infty \Rightarrow g(z) \to$~~

$z = \infty$, $g(z) = 1$

$z = -\infty$, $g(z) = 0$

Decision boundary :

Say, $y=1$ if $h_\theta(x) \ge 0.5$, else $y=0$.



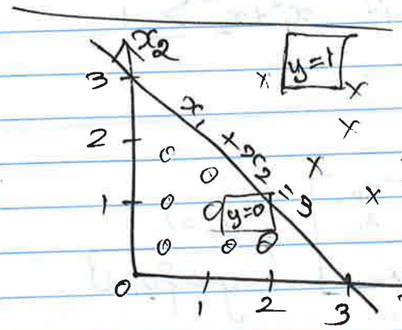
$h_\theta(x) = g(\theta^T x) \to g(z) \ge 0.5$

when $z \ge 0$

$\Rightarrow h_\theta(x) \ge 0.5 \ (y=1)$

when $\theta^T x \ge 0$.

And, $y=0$ when $\theta^T x < 0$.



~~$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$~~

$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

Predict $y=1$ if

$-3 + x_1 + x_2 \ge 0$

(here $\begin{bmatrix}\theta_0\\ \theta_1\\ \theta_2\end{bmatrix} = \begin{bmatrix}-3\\ 1\\ 1\end{bmatrix}$)

Since $y=1$ if $g(z) \ge 0.5$

$\Rightarrow y=1$ if $-3 + x_1 + x_2 \ge 0$

$\Rightarrow y=1$ if $x_1 + x_2 \ge 3$

Here, the line $x_1 + x_2 = 3$ is the decision boundary.

Non-linear decision boundary :

Decision boundary

$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$

$$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

$y = 1$ if $-1 + x_1^2 + x_2^2 \geq 0$

$\Rightarrow y = 1$ if $x_1^2 + x_2^2 \geq 1$

Cost function :

$$x \in \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}, \quad x_0 = 1$$

$y \in \{0, 1\}$

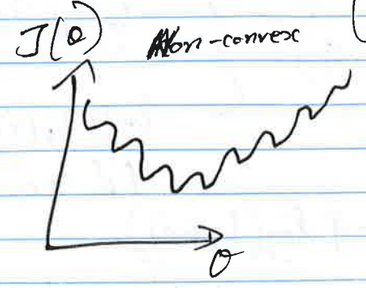
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

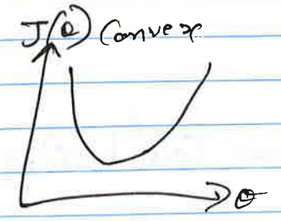
From linear regression,

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{cost}(h_\theta(x), y)$$

$$\text{Cost}(h_\theta(x), y) = \frac{1}{2} (h_\theta(x) - y)^2$$

Cost function for logistic regression is non-convex (no convergence to global minimum)

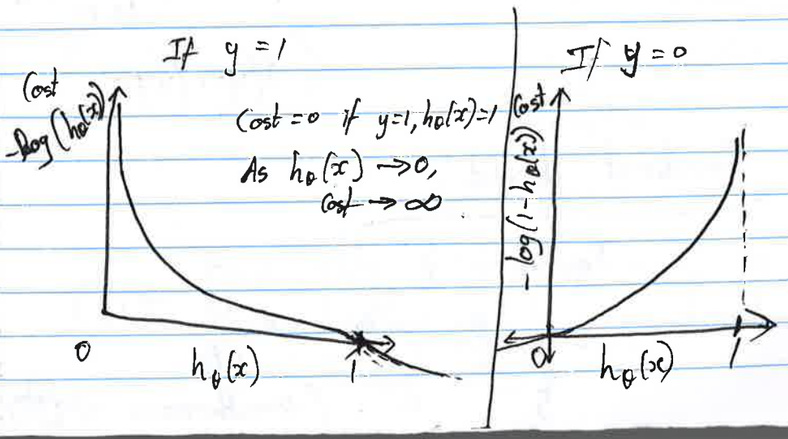




Why is this cost function non-convex?

Logistic regression cost function :

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)), & \text{if } y = 1 \\ -\log(1 - h_\theta(x)), & \text{if } y = 0 \end{cases}$$



Logistic regression Cost function (Simplified):

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}\left(h_\theta(x^i), y^i\right)$$

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{(if } y=1\text{)} \\ -\log(1-h_\theta(x)) & \text{(if } y=0\text{)} \end{cases}$$

$$\Rightarrow \text{Cost}(h_\theta(x), y) = -y\log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log(h_\theta(x^i)) + (1-y^i)\log(1-h_\theta(x^i))\right)$$

(Maximum Likelihood Estimation)

To fit params θ: $\min_\theta J(\theta)$

To predict for a new x: $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

$\left(P(y=1/x;\theta)\right)$

Gradient descent:

Repeat {

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$$

(simultaneous update for all j)

}

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^i) - y^{(i)}\right).x_j^{(i)}$$

$$\theta_j := \theta - \alpha\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right).x_j^{(i)}$$

(simultaneous update all θ_j)

→ Same as the gradient descent for linear regression!!

But, $h_\theta(x)$ here is different.

Advanced Optimization:

Optimization techniques:

- Gradient descent
- Conjugate gradient → No need to manually pick α (learning rate)
- BFGS → Faster than gradient descent
- L-BFGS → More Complex

Essentially employs line search to automatically pick a good value of α (learning rate)

Logistic regression for multi-class classification:

→ One-vs-all (one-vs-rest) classification

(eg) Training set with three classes: A, B, C

↳ Turn this into 3 separate binary classification problems

(Classes: 1, 2, 3)

$h_\theta^{(1)}(x)$, $h_\theta^{(2)}(x)$, $h_\theta^{(3)}(x)$

$h_\theta^{(1)}(x)$ → Binary classification (class 1 vs others)

$h_\theta^{(2)}(x)$ → Class 2 vs others

$h_\theta^{(3)}(x)$ → Class 3 vs others

$h_\theta^{(i)}(x)$ → predict $y=i$

$(P(y=i \mid x;\theta))$

Output:

$\max_i h_\theta^{(i)}(x)$

Regularization:

- Underfitting (High bias)
- Overfitting (High variance)
- "Just right"

↳ Too many hypothesis may fit the training set well (cost function is very low) but fails for new predictions.

Solutions for overfitting:

→ Reduce the no. of features

- Manual selection
- Model selection algorithm

→ Regularization

- Keep all features
- Reduce the magnitude of params θ_j
- Works when there are a lot of features, each contributing a bit.

Penalize the cost function to make some θ_j values really small.

Small values for parameters $\theta_0, \theta_1, \cdots, \theta_n$
- "Simpler" hypothesis
- Less prone to overfitting

We don't know which params to shrink.

~~so regularize~~
(since we don't know which params correspond to which polynomial terms)
So, regularize them all!

Regularized cost function (for linear regression):

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \lambda \sum_{j=1}^{n} \theta_j^2\right]$$

θ_0 needn't be regularized - Since it's a constant term, not much impact.

$$\min_{\theta} J(\theta)$$

$\lambda \rightarrow$ Regularization parameter

What if λ is too large?
↳ Results in underfitting
(As $\theta_0, \theta_1, \cdots, \theta_n$ will be almost zero)

Regularized linear regression:

Cost fn., $J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \lambda \sum_{j=1}^{n} \theta_j^2\right]$

Gradient descent (with regularization):

~~$\theta_0 = \theta_0$~~

$$\theta_0 := \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right).x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta)$$

~~$\Rightarrow \theta_j := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right).x_j^{(i)}$~~

~~$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j}$~~

$$\theta_j := \theta_j - \alpha\left[\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right).x_j^{(i)} + \lambda \cdot \theta_j\right]$$

$$\theta_j := \theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right).x_j^{(i)} - \frac{\alpha\cdot\lambda}{m}\theta_j$$

$$\theta_j := \theta_j\left(1 - \alpha\cdot\frac{\lambda}{m}\right) - \alpha\cdot\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

$\left(1 - \alpha\cdot\frac{\lambda}{m}\right) < 1$ → Reduces θ_j

Everything else is the same.

Normal equation (with regularization)

$X\theta = y$

$X^T X \theta = X^T y$

$\Rightarrow \theta = (X^T X)^{-1} X^T y$

With regularization,

~~$\theta = (X^T X)$~~

$$\theta = \left(X^T X + \lambda \cdot \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

↳ $(n+1) \times (n+1)$ matrix

Even if $X^T X$ is non-invertible ($m <= n$),

$\left(X^T X + \lambda \begin{bmatrix} 0 & & \\ & 1 & \\ & & 1 \end{bmatrix} \right)$ is always invertible

Regularized logistic regression:

$$J(\theta) = -\left[\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log\left(h_\theta(x^{(i)})\right) + (1 - y^{(i)}) \cdot \log\left(1 - h_\theta(x^{(i)})\right) \right] + \frac{\lambda \cdot \sum_{j=1}^{n} \theta_j^2}{2m}$$

Gradient descent:

$$\theta_j := \theta_j \left(1 - \alpha \cdot \frac{\lambda}{m}\right) - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

Week 4:

Neural networks: Representation:

Non-linear hypothesis:

↳ - For training set with lot of features, polynomial feature mapping will generate too many features.

$n \rightarrow$ number of features

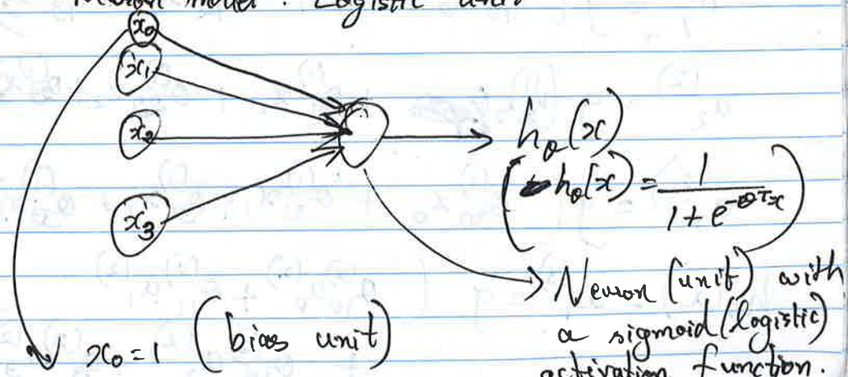
Number of quadratic features (feature mapping) $= O(n^2)$ $\left(\simeq \frac{n^2}{2}\right)$

↳ ${}^{n}C_2$

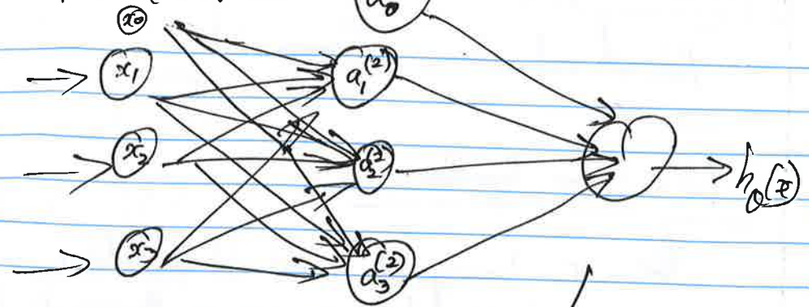
If number of features is large, the total no. of generated features is way too large even for quadratic hypothesis.

Model Representation:

Neuron model: Logistic unit



Neural network : $a_0^{(2)}$ (bias unit)



$a_i^{(j)}$ → activation of unit i in layer j

$\theta^{(j)}$ → Matrix of weights controlling function mapping from layer j to layer $(j+1)$.

g → sigmoid function (activation function)

$$a_1^{(2)} = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right)$$

$$a_2^{(2)} = g\left(\theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3\right)$$

$$a_3^{(2)} = g\left(\theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3\right)$$

$$h_\theta(x) = a_1^{(3)} = g\left(\theta_{10}^{(2)}a_0^{(2)} + \theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}a_2^{(2)} + \theta_{13}^{(2)}a_3^{(2)}\right)$$

Here, $\theta^{(1)} \in \mathbb{R}^{3\times 4}$, $\theta^{(2)} \in \mathbb{R}^{1\times 4}$

Network has s_j units in layer j and s

s_j units in layer j & s_{j+1} units in layer $j+1$ $\Rightarrow$ $\theta^{(j)}$ will be a $s_{j+1} \times (s_j+1)$ dimensional matrix

Mapping from layer j to layer $(j+1)$

Additional 1 column for the bias unit.

Forward prop : Vectorized implementation

$$a_1^{(2)} = g(z_1^{(2)}) \quad \left(z_1^{(2)} = \theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 \ldots\right)$$
$$a_2^{(2)} = g(z_2^{(2)})$$
$$a_3^{(2)} = g(z_3^{(2)})$$

$$z_1^{(2)} = \theta_1^{(1)} x, \quad x$$ (Vectorized)

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}; \quad \boxed{z^{(2)} = \theta^{(1)} x}$$

$$\Rightarrow a^{(2)} = g(z^{(2)})$$

$$\boxed{a^{(2)} = g(\theta^{(1)} \cdot x) \quad (\text{or}) \quad a^{(2)} = g(\theta^{(1)} \cdot a^{(1)})} \quad (a^1 = x).$$

Now, add the bias unit

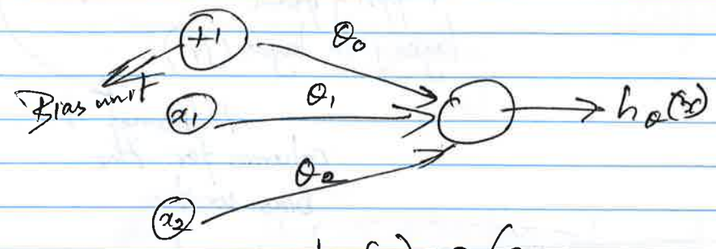
Add $a_0^{(2)} = 1$.

$z^{(3)} = \Theta^{(2)} \cdot a^{(2)}$

$$h_\Theta(x) = a^{(3)} = g(z^{(3)}) = g(\Theta^{(2)} \cdot a^{(2)})$$

Example: AND

$x_1, x_2 \in \{0, 1\}$; $y = x_1$ AND x_2

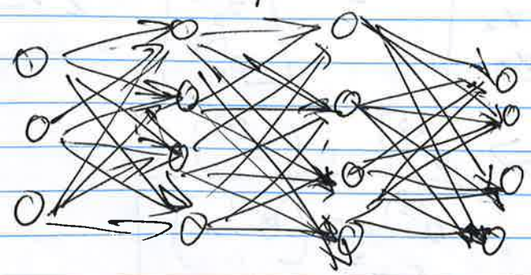


$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$

Multi-class classification neural network:

→ One vs all

→ multiple output units



$h_\Theta(x) \in \mathbb{R}^4$

Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots$

$y^{(i)}$ one of $\begin{bmatrix}1\\0\\0\\0\end{bmatrix} \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

Class 1, Class 2, Class 3, Class 4

Week 5

Neural n/w (classification):

$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

↳ Training set

L = total no. of layers in NN

s_l = no. of units (not including the bias unit) in layer l.

Cost function:

Logistic regression cost fn.:

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log\left(1 - h_\Theta(x^{(i)})\right) \right] + \frac{\lambda}{2m} \sum_{j=1} \Theta_j^2.$$

Cost function of NN:

$K \Rightarrow$ no. of classes (i.e., no. of output units)

$h_\Theta(x) \in \mathbb{R}^K$

$(h_\Theta(x))_i = i^{th}$ output

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\Theta(x^{(i)})\right)_k + \left(1 - y_k^{(i)}\right)\log\left(1 - h_\Theta(x^{(i)})\right)_k\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2$$

Backpropagation:

$\min_\Theta J(\Theta)$

Need code to compute:

$J(\Theta)$ & $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Gradient computation: Backpropagation algorithm:

Intuition: $\delta_j^{(l)} \rightarrow$ error of unit j in layer l.

$\rightarrow$ (i.e.) error in $a_j^{(l)}$.

For each output unit (layer L)

$\delta_j^L = a_j^L - y_j$

$\underbrace{\quad}_{(h_\Theta(x))_j}$

$\Rightarrow$ Vectorized: $\boxed{\delta^L = a^L - y}$

$\downarrow$

Error in the output layer.

Now compute the errors in the previous layers.

$$\delta^{(L-1)} = \left(\left(\Theta^{(L-1)}\right)^T \cdot \delta^{(L)}\right) .* g'\left(z^{(L-1)}\right)$$

$\downarrow$ Element-wise multiplication

NN with 4 layers:

$\delta^{(4)} = a^{(4)} - y$

$\delta^{(3)} = \left(\Theta^{(3)}\right)^T \delta^{(4)} .* g'\left(z^{(3)}\right)$ ($g'(z^{(3)}) = a^{(3)} .* (1 - a^{(3)})$)

$\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} .* g'\left(z^{(2)}\right)$ ($g'(z^{(2)}) = a^{(2)} .* (1 - a^{(2)})$)

(No $\delta^{(1)}$)

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

(Can be proven)

(Ignoring λ here ($\lambda = 0$): No regularization)

Backpropagation algorithm:

Training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

Set $\Delta_{ij}^{(l)} = 0$ (for all i, j, l)

(Used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

For $i = 1$ to m (each training set $(x^{(i)}, y^{(i)})$):

→ Set $a^{(1)} = x^{(i)}$

→ Perform forward prop to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$.

→ Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

→ Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

→ $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$.

↳ [Vectorized: $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$]

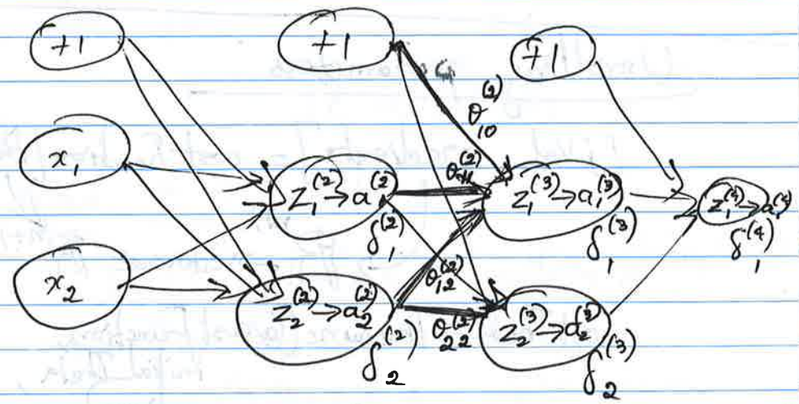
→ $D_{ij}^{(l)} := \frac{1}{m} \left(\Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \right)$ if $j \neq 0$

→ $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$ (No regularization for the bias unit).

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}.$$

Back prop (Intuition):

forward prop:



$$z_1^{(3)} = \Theta_{10}^{(2)} \times 1 + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)}$$

Cost function (for 1 training set, with one-class (one output unit), i.e., $k=1$, and no regularization, i.e., $\lambda = 0$)

$$\text{cost}(i) = y^{(i)} \log\left(h_\Theta(x^{(i)})\right) + (1 - y^{(i)}) \log\left(1 - h_\Theta(x^{(i)})\right)$$

$\delta_j^{(l)}$ → error of cost for $a_j^{(l)}$ (unit j in layer l).

$$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \left(\text{cost}(i)\right) \quad (\text{for } j \geq 0)$$

$$\delta_1^{(4)} = y_1^{(i)} - a_1^{(4)}$$

$$\delta_2^{(3)} = \Theta_{12}^{(2)} \cdot \delta_1^{(3)} + \Theta_{22}^{(2)} \cdot \delta_2^{(3)}$$

$$\delta_2^{(3)} = \Theta_{12}^{(3)} \cdot \delta_1^{(4)}$$

Unrolling parameters:

[jVal, gradients] = costFunction(thetaVec)

→ $\mathbb{R}^{n+1}$ →vectors← $\mathbb{R}^{n+1}$

optTheta = fminunc(@costFunction, initialTheta, options)

NN (4 layers):

$\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ — matrices of params

$D^{(1)}, D^{(2)}, D^{(3)}$ — gradient matrices

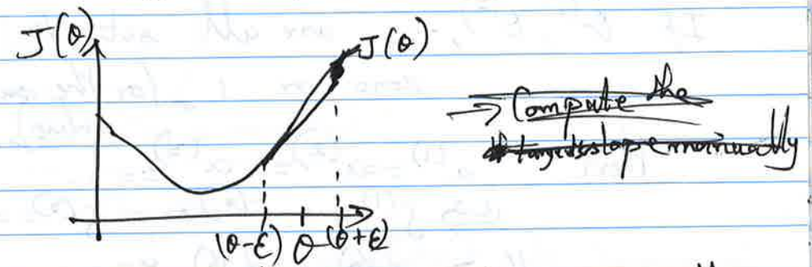
Unroll them into vectors for above

(column-major ~~row-major~~)

Gradient Checking:

→ Numerical estimation of gradients

→ To verify back prop works as expected.



→ Compute the ~~tangent~~ slope numerically

↳ Manually compute the slope of the ~~line~~ tangent

$$\frac{d}{d\theta} J(\theta) \simeq \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$$

(ε is typically 10^{-4})

When θ is a vector:

$\theta \in \mathbb{R}^n$ (after unrolling $\theta^{(1)}, \theta^{(2)}, \ldots$)

$\theta = [\theta_1, \theta_2, \ldots, \theta_n]$

$$\frac{\partial}{\partial \theta_1} J(\theta) \simeq \frac{\left(J(\theta_1+\epsilon, \theta_2, \theta_3, \ldots, \theta_n) - J(\theta_1-\epsilon, \theta_2, \theta_3, \ldots, \theta_n)\right)}{2\epsilon}$$

$$\frac{\partial}{\partial \theta_2} J(\theta) \simeq \frac{\left(J(\theta_1, \theta_2+\epsilon, \theta_3, \ldots, \theta_n) - J(\theta_1, \theta_2-\epsilon, \theta_3, \ldots, \theta_n)\right)}{2\epsilon}$$

⋮

Random Initialization :

Initial value of θ

Zero initialization doesn't work.

If $\theta^{(1)}, \theta^{(2)}, \ldots$ are all set to zero or 1 (or the same value),

then $a^{(1)} = a^{(2)} = a^{(3)} = \ldots$

$\hookrightarrow \delta^{(1)} = \delta^{(2)} = \delta^{(3)} = \ldots$

$\hookrightarrow \frac{\partial J(\theta)}{\partial \theta_{01}^{(1)}} = \frac{\partial J(\theta)}{\partial \theta_{02}^{(1)}} = \ldots$

$\hookrightarrow$ After 1 iteration $\theta^{(1)} = \theta^{(2)} = \theta^{(3)} = \ldots$

Use random initialization

↳ Symmetry ~~break~~ breaking

Initialize each θ_{ij}^{l} to random value in $[-\epsilon, \epsilon]$

Steps:
→ Pick NN architecture
→ Randomly initialize weights
→ Compute $h_\theta(x)$ (Forward prop)
→ Compute cost function $J(\theta)$
→ Compute gradients $\frac{\partial}{\partial \theta_{jk}^{(l)}} J(\theta)$ (Back prop)
→ Compare backprop with numerical gradient checking and verify it works
→ Minimize $J(\theta)$ over θ using gradient descent or advanced optimization

NN cost function need not be convex

↳ It's okay to converge to local minima.

Non-convex

Week 6

Evaluating a hypothesis :

- Overfitting
  ↳ Split dataset into training set & test set

Misclassification error (0/1 misclassification error)

$$err(h_\theta(x), y) = \begin{cases} 1, & \text{if } h_\theta(x) >= 0.5, y = 0 \\ & \text{or if } h_\theta(x) < 0.5, y = 1 \\ 0, & \text{otherwise} \end{cases}$$

$$\text{Test error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_\theta(x_{test}^{(i)}, y_{test}^{(i)}))$$

Model Selection :

- Picking a function for $h_\theta(x)$
- What degree of polynomial to pick?

Say, pick a ~~dg~~ degree (d) based on which minimizes $J_{test}(\theta)$.

↓

Not fair as $J_{test}(\theta)$ is likely to be an optimistic estimate of generalization error (i.e., d is chosen to fit the test set)

Solution:

Split dataset into

Train set

Test set

Cross Validation (CV) set

- Bunch of models
- Pick the best performing model (lowest cost function or error) on the cross validation set (i.e., pick the model with lowest cross validation error)

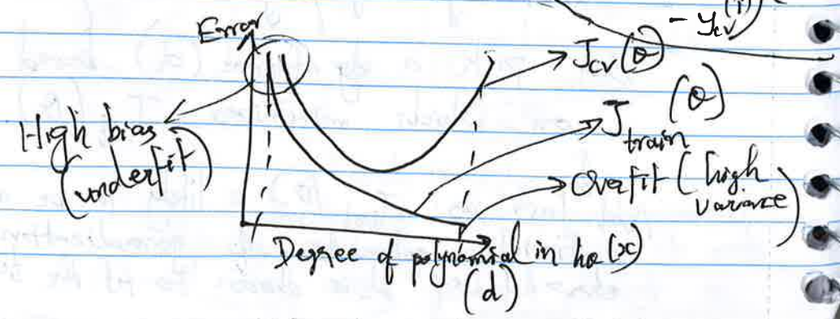
Bias and Variance:

High Bias - underfitting

High Variance - overfitting

Training error: $J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Cross-validation error: $J_{cv}(\theta) = \frac{1}{2m} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$



Underfit (bias):

$J_{train}(\theta)$ will be high

$J_{cv}(\theta) \approx J_{train}(\theta)$

Overfit (variance):

$J_{train}(\theta)$ will be low

$J_{cv}(\theta) \gg J_{train}(\theta)$

Regularization and bias/variance:

Large λ (regularization param) $\rightarrow$ Underfit

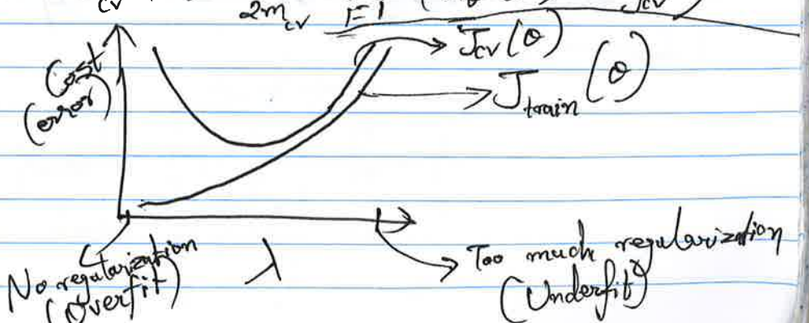
Small $\lambda \rightarrow$ Overfit

Choosing λ correctly:

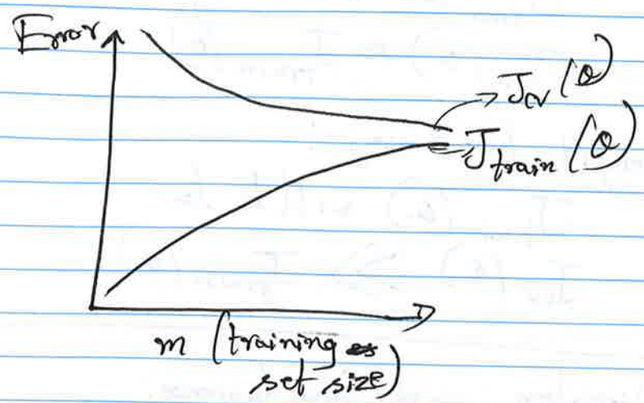
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^{m} \theta_j^2$$

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

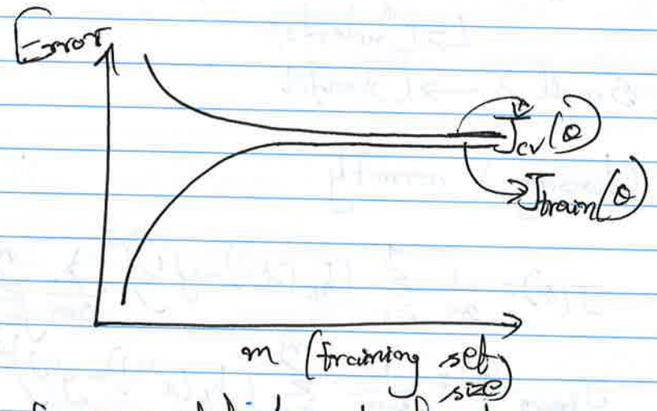
$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$



Learning Curves:

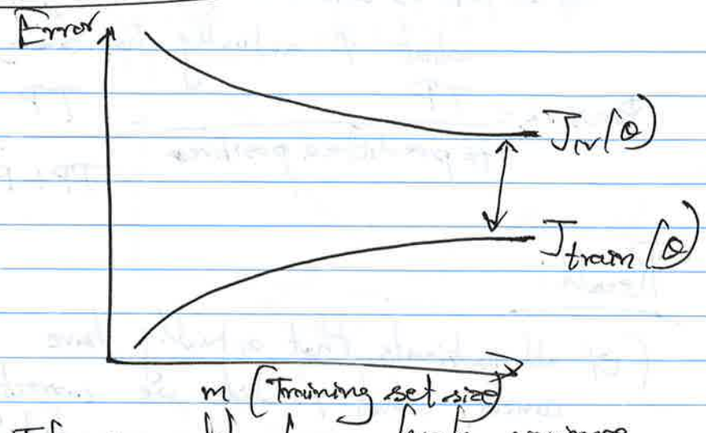


High bias (Underfit):



If a model has high bias, more training data won't help.

High variance (Overfit):



If a model has high variance, more training data might help.

Error metrics for skewed classes:

Skewed class → Training set has skew in output labels

99.5% $y=0$, 0.5% $y=1$

Precision/Recall:

$y=1$ in presence of rare class that we want to predict.

		Actual class	
		1	0
Predicted class	1	TP	FP
	0	FN	TN

Precision:

(Of all patients where we predicted $y=1$, what % actually has cancer)

$$\text{Precision} = \frac{TP}{\#\text{predicted positives}} = \frac{TP}{TP+FP}$$

Recall:

(Of all patients that actually have cancer, what % did we correctly detect)

~~Recall = True positive~~

$$\text{Recall} = \frac{TP}{\#\text{actual positives}} = \frac{TP}{TP+FN}$$

Trading off precision and recall:

→ Say, predict cancer only if highly confident.

Predict 1 if $h_\theta(x) >= 0.7$

Predict 0 if $h_\theta(x) < 0.7$

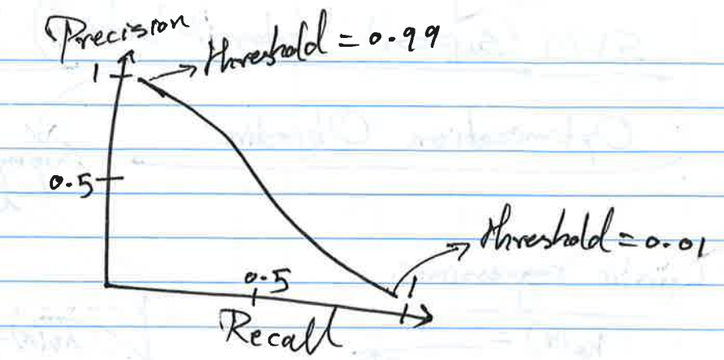
↳ Higher precision, lower recall.

→ Say, we want to avoid missing too many cases of cancer (avoid false negatives)

Predict 1 if $h_\theta(x) >= 0.3$, 0 otherwise

↳ Lower precision, higher recall

Predict 1 if $h_\theta(x) >= \text{threshold}$



F_1 score (F score):

- Comparing precision / recall numbers

	Precision (P)	Recall (R)
Algo 1	0.5	0.4
Algo 2	0.7	0.1
Algo 3	0.02	1.0

$$F_1 \text{ score} = 2\frac{PR}{(P+R)}$$

$P=0$ or $R=0 \Rightarrow$ F-score $= 0$

$P=1$ and $R=1 \Rightarrow$ F-score $= 1$

Week 7

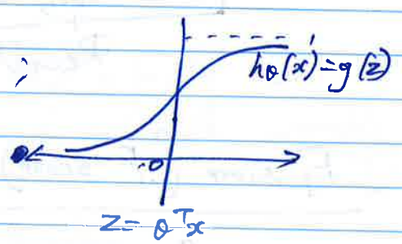
SVM (Support Vector Machine):

Optimization Objective:

Supervised learning

Logistic regression:

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$;



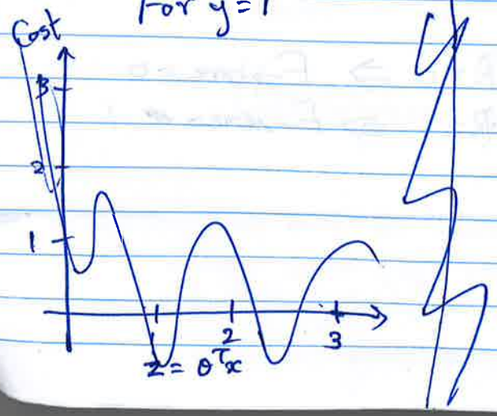
If $y=1$, we want $h_\theta(x) \simeq 1$, $\theta^T x \gg 0$

If $y=0$, we want $h_\theta(x) \simeq 0$, $\theta^T x \ll 0$

$$Cost = -y \log h_\theta(x) - (1-y) \log(1-h_\theta(x))$$

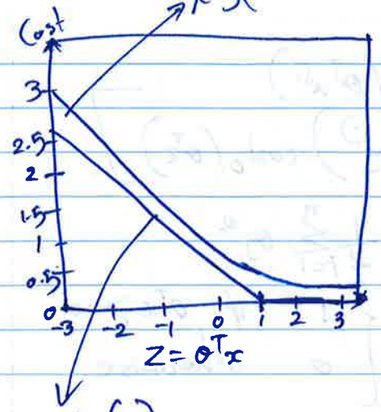
$$Cost = -y \log\left(\frac{1}{1+e^{-\theta^T x}}\right) - (1-y) \log\left(1 - \frac{1}{1+e^{-\theta^T x}}\right)$$

For $y=1$ ↓ For $y=0$ ↓



$y=1$:

$$-\log\left(\frac{1}{1+e^{-\theta^T x}}\right)$$

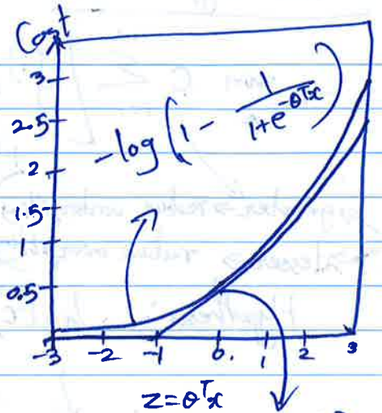


$Cost_1(z)$

$cost_1(z) = \max(0, k(1-z))$

$y=0$:

$$-\log\left(1 - \frac{1}{1+e^{-\theta^T x}}\right)$$



$Cost_0(z)$

$cost_0(z) = \max(0, k(1+z))$

SVM cost function:

$$\min_\theta \; \cancel{\frac{1}{m}} C \sum_{i=1}^{m} \left(y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)}) \right) + \cancel{\frac{1}{2m}} \sum_{i=1}^{n} \theta_j^2$$

→ No division by m

→ Regularization parameter moved

(Instead of $A + \lambda B$, use $CA + B$.

$C = \frac{1}{\lambda}$)

SVM hypothesis:

$$\rightarrow \min_{\theta} C \sum_{i=1}^{m} \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right]$$

$$\left(C \rightarrow \begin{cases} \text{greater} \rightarrow \text{reduce underfitting} \\ \text{lesser} \rightarrow \text{reduce overfitting} \end{cases} \right) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

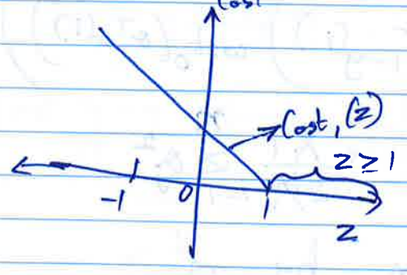
Hypothesis: $h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T x >= 0 \\ 0 & \text{otherwise} \end{cases}$

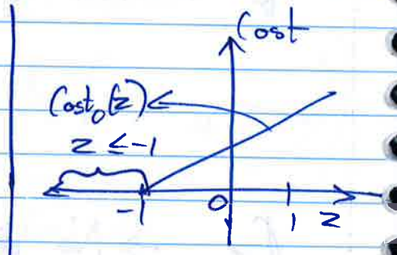
↓

Does not predict the probability like logistic regression

(Discriminant function)

SVM → Large Margin Classifier:

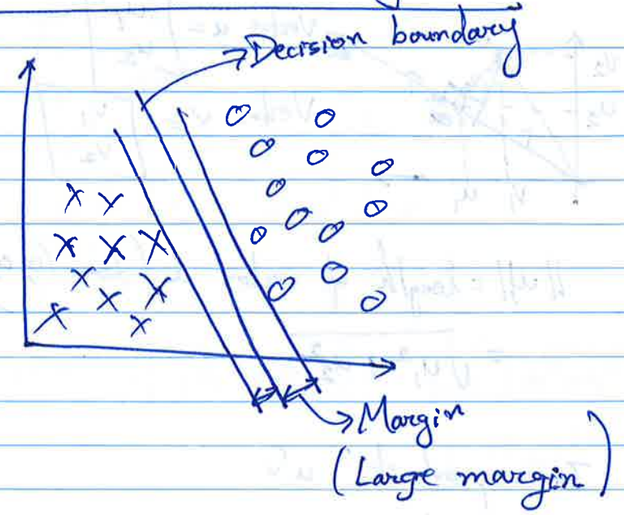




If $y=1$, we want $\theta^T x \geq 1$ (not just $\theta^T x \geq 0$)

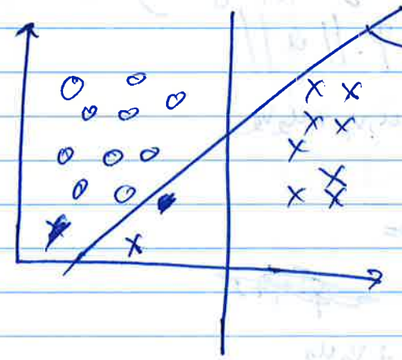
If $y=0$, we want $\theta^T(x) \leq -1$ (not just $\theta^T x < 0$)

SVM Decision Boundary:



(Large margin)

Large margin classifier in presence of outliers:

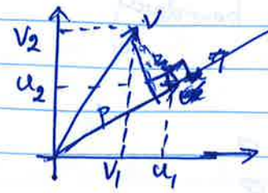


Very large value of C (regularization param)

↓

overfitting

Vector Inner product:



Vector $u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$

Vector $v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$

$\|u\|$ = length of vector u (from $(0,0)$)

$= \sqrt{u_1^2 + u_2^2}$

Inner product $u^T v$

$p \rightarrow$ length of projection of v onto u.

$\rightarrow$ can be signed (negative)

negative if angle is obtuse

$u^T v = p \cdot \|u\|$

$= u_1 v_1 + u_2 v_2$

(or)

$u^T v = v^T u$

$= v_1 u_1 + v_2 u_2$

SVM Decision boundary:

$$\min_\theta \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

s.t.,

$\theta^T x^{(i)} \geq 1$ if $y^{(i)} = 1$

$\theta^T x^{(i)} \leq -1$ if $y^{(i)} = 0$

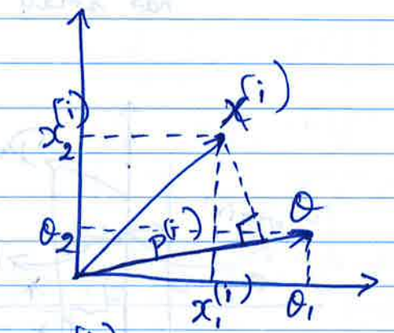
Simplification: $\theta_0 = 0$, $n = 2$

$$\Rightarrow \min_\theta \frac{1}{2}(\theta_1^2 + \theta_2^2) = \frac{1}{2}\left(\sqrt{\theta_1^2 + \theta_2^2}\right)^2$$

$$= \frac{1}{2}\|\theta\|^2$$

$\theta^T x^{(i)} = ?$

$u^T \quad v$



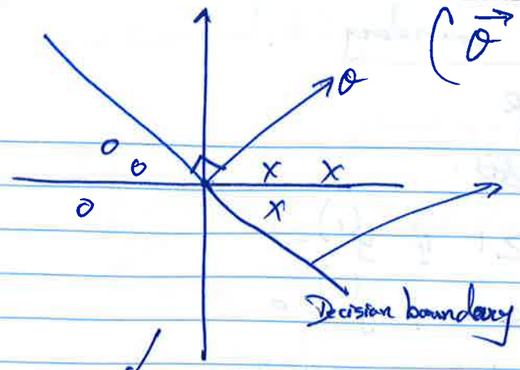
$p^{(i)} \rightarrow$ Projection of $x^{(i)}$ onto θ

$\theta^T x^{(i)} = p^{(i)} \cdot \|\theta\|$

$= \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}$

$$\Rightarrow \min_\theta \frac{1}{2}\|\theta\|^2 \quad \text{s.t.,} \quad p^{(i)} \cdot \|\theta\| \geq 1 \text{ if } y^{(i)} = 1,$$

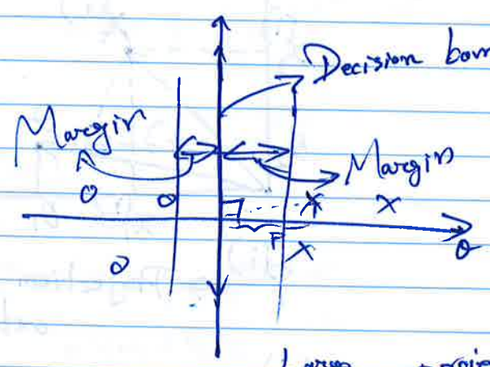
$p^{(i)} \cdot \|\theta\| \leq -1$ if $y^{(i)} = 0$



($\vec{\theta}$ is 90° to the decision boundary)

Not a great decision boundary (no large margin)

Projections are small
↳ $p^{(i)} \cdot \|\theta\| \geq 1$ or $p^{(i)} \cdot \|\theta\| \leq -1$ won't be true if $p^{(i)}$ (projection) has small magnitude.



Large projections of $x^{(i)}$ onto θ

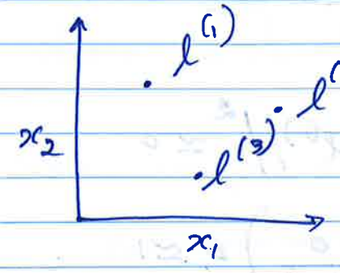
Non-linear decision boundary:

Option 1: Polynomial features.
(Predict $y = 1$ if $\theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2 + \dots \geq 0$.)

or

Kernel:

Kernel:



Given x, compute new features depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$.

Given x: $f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$

$f_2 = \text{similarity}(x, l^{(2)}) = \exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right)$

$f_3 = \text{similarity}(x, l^{(3)}) = \dots$

↳ kernel (this similarity is a Gaussian kernel)

Kernels and Similarity:

$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$

$= \exp\left(-\frac{\sum_{j=1}^{n} (x_j - l_j^{(1)})^2}{2\sigma^2}\right)$

If $x \simeq l^{(1)}$:

↳ $\|x - l^{(1)}\|^2 \simeq 0$

$\Rightarrow f_1 \simeq \exp\left(\frac{-0}{2\sigma^2}\right) \simeq 1$

If x is far from $l^{(1)}$:

$f_1 \simeq \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \simeq 0$

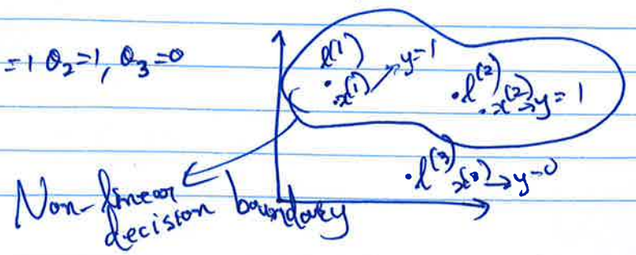
New features → f_1, f_2, f_3

Predict $y = 1$ when

$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Example

$\theta_0 = -0.5, \theta_1 = 1, \theta_2 = 1, \theta_3 = 0$



Choosing the landmarks:

How do you choose $l^{(1)}, l^{(2)}, l^{(3)}$?

SVM with kernels:

Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$,

choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$.

Given example x:

$f_1 = \text{similarity}(x, l^{(1)})$

$f_2 = \text{similarity}(x, l^{(2)})$

$\vdots$

$f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}$, $f_0 = 1$

↳ Feature vector

For training example $(x^{(i)}, y^{(i)})$:

$f_1^{(i)} = \text{sim}(x^{(i)}, l^{(1)})$ ↳ similarity

$f_2^{(i)} = \text{sim}(x^{(i)}, l^{(2)})$

$\vdots$ ← $f_i^{(i)} = \text{sim}(x^{(i)}, l^{(i)}) = \exp\left(\frac{-0}{2\sigma^2}\right) = 1$

$f_m^{(i)} = \text{sim}(x^{(i)}, l^{(m)})$

$f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ f_2^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$; $f_0^{(i)} = 1$

↓ Feature vector for $x^{(i)}$

Given x, compute features $f \in \mathbb{R}^{m+1}$

Predict $y=1$ if $\theta^T f \geq 0$

Hypothesis:

$\theta^T f = \theta_0 f_0 + \theta_1 f_1 + \ldots + \theta_m f_m$

$\theta \in \mathbb{R}^{m+1}$

Training:

$$\min_{\theta} C \sum_{i=1}^{m} \left[y^{(i)} \text{cost}_1(\theta^T f^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T f^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

(Note, no. of features f = no. of training examples (+1 for bias)).

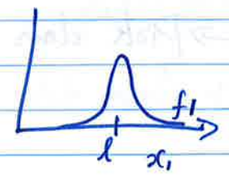
SVM parameters:

$C (= \frac{1}{\lambda})$ → large C → low bias, high variance (overfit)
→ small C → high bias, low variance (underfit)

σ^2 (for Gaussian kernel in similarity matrix)

Large σ^2 : ↓ Underfitting

Small σ^2 : ↓ Overfitting



Using an SVM:

→ Choose param C

→ Choose kernel (similarity function)

→ No kernel (linear kernel): Predict $y=1$ if $\theta^T x \geq 0$

→ Gaussian kernel: $f_i = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$, where $l^{(i)} = x^{(i)}$

(Need to choose σ^2).

(Perform feature scaling before using the kernel)

→ Polynomial kernels: $k(x, l) = (x^T l)^2$, $(x^T l)^3$, $(x^T l + 1)^3$, $(x^T l + 5)^4$

Multiclass classification with SVM:

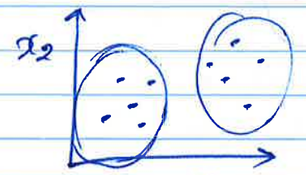
→ One vs all method
→ Train K SVMs, each for one of the K classes.
→ Pick class i with largest $(\theta^{(i)})^T x$.

Logistic Regression vs SVM:

n → no. of features
m → no. of training examples
→ large n (relative to m):
↳ Use logistic regression or SVM w/o kernel (linear kernel)
→ small n, intermediate m:
↳ SVM w/ Gaussian kernel
→ small n, large m:
↳ Add more features, then use logistic regression or SVM w/o kernel.

Week 8:

Unsupervised Learning:



- Clustering

Training set: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$
No labels $y^{(i)}$

Clustering: K-means algorithm:

Input:
- K (number of clusters)
- Training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

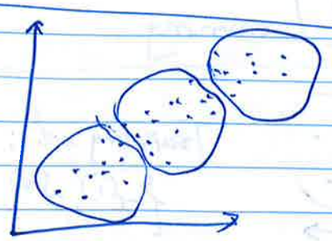
$x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)

Algo:
- Randomly initialize K cluster ~~centroids~~ centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$
- Repeat {

Cluster assignment step:
for $i = 1$ to m
$c^{(i)}$:= index (from 1 to k) of cluster centroid closest to $x^{(i)}$

Move centroid step:
for $k = 1$ to K
μ_k = average (mean) of points assigned to cluster k.
}

K-means for non-separated clusters:



Optimization Objective:

$c^{(i)}$ → cluster for example $x^{(i)}$

μ_k → centroid of cluster k.

$\mu_{c^{(i)}}$ → centroid of cluster to which $x^{(i)}$ has been assigned

Cost fn.:

$$J(c^{(1)}, c^{(2)}, \dots, c^{(m)}, \mu_1, \mu_2, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - \mu_{c^{(i)}} \|^2$$

$$\min_{c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$$

Random Initialization:

- Random initialization of cluster centroids at startup
  - → Should have $K < m$
  - → Randomly pick K training examples
  - → Set $\mu_1, \mu_2, \dots, \mu_K$ equal to these K examples.

Note:
Local optima is possible based on centroid initialization.

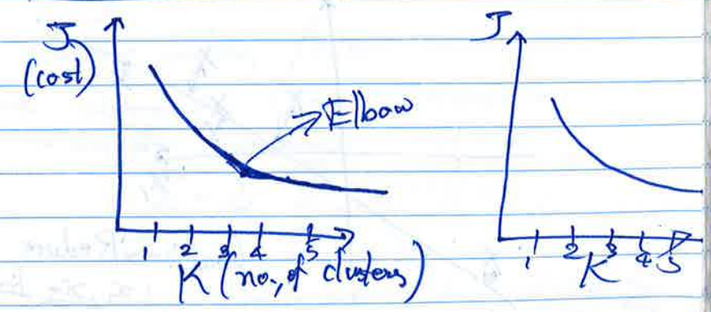
↓

Solution: → Try K-means multiple times with different random centroid initialization.
→ Pick the one with the least cost.

Choosing K (no. of clusters):

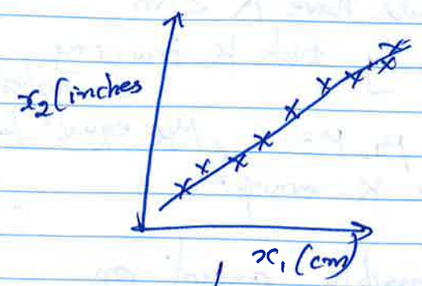
→ Manually, visualization, etc.

→ Elbow method:



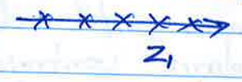
Dimensionality Reduction:

- Data Compression: → Reduce unnecessary or redundant features



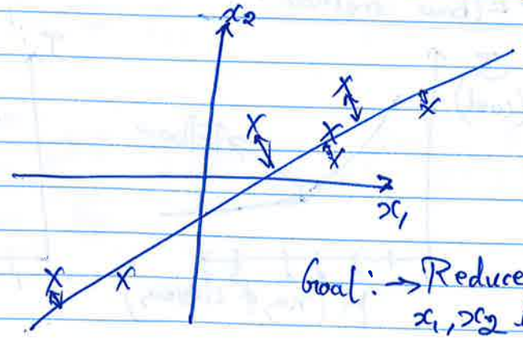
→ Reduce x_1 & x_2 into a new feature z_1

→ Project onto x_1.



- Data Visualization:

PCA (Principal Component Analysis):



$x \in \mathbb{R}^2$

Goal: → Reduce 2 feature x_1, x_2 to 1 feature z_1

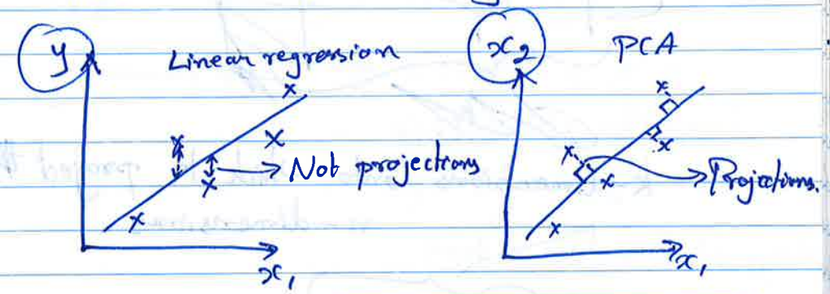
→ Find a line to project the points onto

PCA → find a ~~line~~ line such that the projection error (~~dist.~~ sum of squares of projection distance) is minimal.

Reduce n-dimensions to k-dimensions:

→ Find k vectors $u^{(1)}, u^{(2)}, \dots, u^{(k)}$ onto which to project the data, so as to minimize the projection error.

PCA is NOT linear regression:



PCA algorithm:

→ How to find the k dimensions

→ Then, how to find the projected new features for each training example

→ Data preprocessing:

Training set: $x^{(1)}, x^{(2)}, \dots, x^{(m)}$

~~Scaling~~ Feature scaling / Mean normalization

→ Reduce n-dimensions to k-dimensions

→ Compute covariance matrix

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} \left(x^{(i)}\right)^T \left(x^{(i)}\right)$$

(Σ: covariance matrix)

→ Compute eigen vectors of matrix Σ

$[U, S, V] = \text{svd}(\Sigma)$ (Singular Value Decomposition)

Covariance matrix Σ

$\downarrow$

$n \times n$ matrix.

Covariance → How much two random variables change together.

$[U, S, V] = svd(\Sigma)$ → $n \times n$

$U = [u^{(1)} \; u^{(2)} \; \cdots \; u^{(k)} \; \cdots \; u^{(n)}]$ (k columns)

k-dimensions onto which to project the n-dimensions.

$U = [u^{(1)} \; u^{(2)} \; \cdots \; u^{k} \; \cdots \; u^{(n)}] \in \mathbb{R}^{n \times n}$

$x \in \mathbb{R}^n$

$x^{(i)} \in \mathbb{R}^n$

$x \in \mathbb{R}^n \rightarrow z \in \mathbb{R}^k$

$[u^{(1)} \; u^{(2)} \; \cdots \; u^{(k)}] \in \mathbb{R}^{n \times k}$

$\downarrow$

U_{reduce}

$z^{(i)} = U_{reduce}^T \cdot x^{(i)}$

$(k \times n) \cdot (n \times 1) \Rightarrow k \times 1 \Rightarrow$ k-dimensions.

$z^{(i)} \in \mathbb{R}^k$

Summary of PCA algo:

1.) $X \in \mathbb{R}^{m \times n}$

2.) $Sigma = \frac{1}{m} \cdot X^T X$ (covariance matrix)

3.) $[U, S, V] = svd(Sigma)$

4.) $U_{reduce} = U[:, 1:k]$ → First k columns of U → $\mathbb{R}^{n \times k}$

5.) $Z = X \cdot U_{reduce}$ → $\mathbb{R}^{m \times k}$ → $\mathbb{R}^{n \times k}$

Reconstruction from Compressed Representation:

Given $z \in \mathbb{R}^k$, get back $x \in \mathbb{R}^n$.

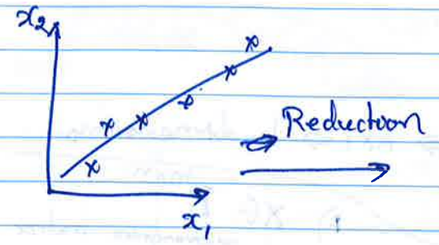
(approx., x).

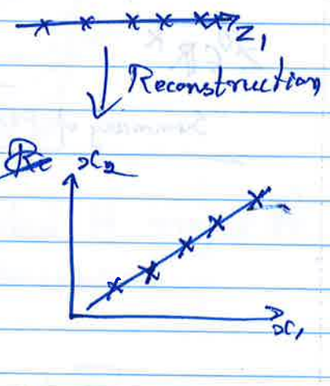
$x_{approx} = U_{reduce}^T \cdot z$

$X_{approx} = Z \cdot U_{reduce}^T$

$m \times n$ $\quad m \times k$ $\quad k \times n$

Approx reconstruction of X





Goal of PCA:

$$\min \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|$$

Choosing the no. of principal components (k):

Choosing k:

PCA → tries to minimize mean square projection error.

$$\left(\frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \| \right)$$

Total variation $= \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} \|^2$

↙ Covariance between $x^{(i)}$ and origin.

Choose the smallest k such that

$$\frac{\frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2}{\frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} \|^2} \leq 0.01 \quad (1\%)$$

→ 99% of variance is retained.

$[U, S, V] = svd(Sigma)$

↓

$n \times n$ diagonal matrix.

$$S = \begin{bmatrix} S_{11} & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & S_{22} & 0 & \cdots & & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & \cdots & & & 0 & S_{nn} \end{bmatrix}_{n \times n}$$

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.01$$

Summary of choosing k (no. of principal components):

- $[U, S, V] = svd(Sigma)$

Pick smallest k for which

$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{m} S_{ii}} \geq 0.99$$

(99% of variance is retained)

Applying PCA:

- Speedup supervised learning

$(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$

$x^{(i)} \in \mathbb{R}^{10000}$

$\Rightarrow$ Slow learning (too many features)

Extract inputs: $x^{(1)}, \dots, x^{(m)}$

$\downarrow$ PCA

$z^{(1)}, \dots, z^{(m)}$

New training set:

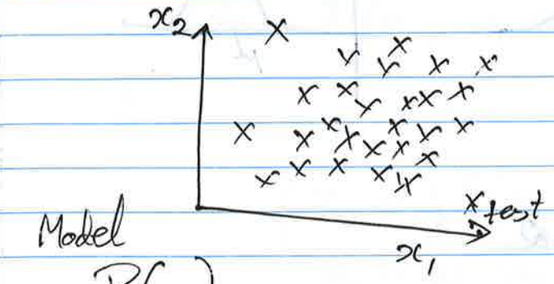
$(z^{(1)}, y^{(1)}), \dots, (z^{(m)}, y^{(m)})$

Note: Do NOT use PCA just to reduce overfitting

Week 9:

Anomaly detection:

Dataset: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$



Model $P(x)$

$P(x_{test}) < \epsilon \rightarrow$ Anomaly

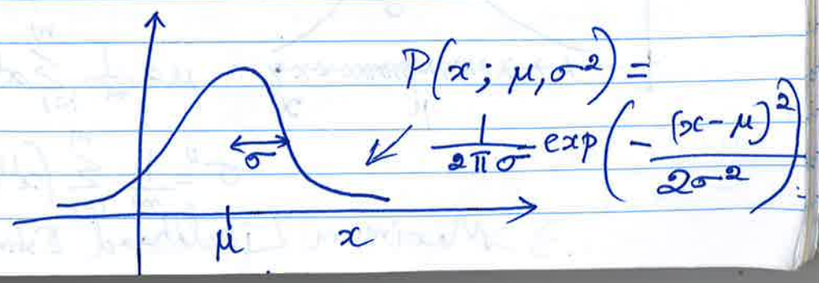
$P(x_{test}) >= \epsilon \rightarrow$ OK

Gaussian (Normal) distribution:

$x \in \mathbb{R}$, $x \sim \mathcal{N}(\mu, \sigma^2)$

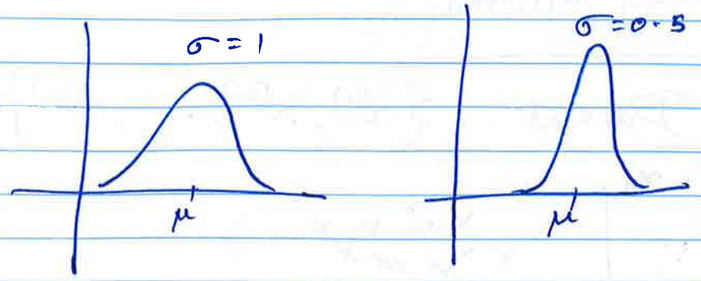
"distributed as"

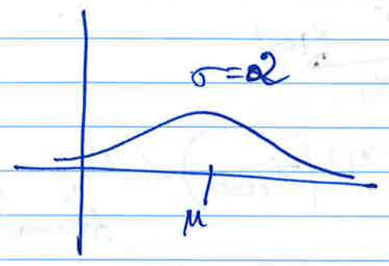
$\mu \rightarrow$ mean, $\sigma^2 \rightarrow$ variance



$$P(x; \mu, \sigma^2) = \frac{1}{2\pi\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

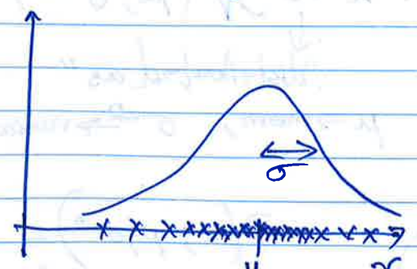
Examples:





Parameter estimation:

Dataset: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$,
$x^{(i)} \in \mathbb{R}$



$x^{(i)} \sim \mathcal{N}(\mu, \sigma^2)$

$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$

$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)^2$

Maximum Likelihood Estimates

Density estimation:
(Anomaly detection)

→ Training set: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

→ For each example $x \in \mathbb{R}^n$

$$P(x) = P(x_1; \mu_1, \sigma_1^2) \cdot P(x_2; \mu_2, \sigma_2^2) \cdots P(x_n; \mu_n, \sigma_n^2)$$

$$\begin{pmatrix} x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2) \\ x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2) \\ \vdots \\ x_n \sim \mathcal{N}(\mu_n, \sigma_n^2) \end{pmatrix}$$

$P(x) = \prod_{j=1}^{n} P$ ~~P_x~~ $\quad P(x) = \prod_{j=1}^{n} P(x_j; \mu_j, \sigma_j^2)$

Anomaly detection algorithm:

1.) Choose features that might be indicative of anomalies

2) Fit params $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j \quad ; \sigma_j^2 = \frac{1}{m} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2$$

3.) Given new example x, compute $p(x)$:

$P(x) = \prod_{j=1}^{n} P(x_j; \mu_j, \sigma_j^2)$

$= \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$

Anomaly if $P(x) < \epsilon$.

Developing & Evaluating Anomaly Detection System:

Training set → non-anomalous

Cross-validation } → few anomalous examples.
Test-set }

→ Fit model $p(x)$ on training set

→ On cross-validation / test example x, predict

$$y = \begin{cases} 1 & \text{if } p(x) < \epsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) >= \epsilon \text{ (normal)} \end{cases}$$

Evaluation metrics:
- Precision / Recall
- F_1-score

Cross-validation set to choose param ϵ.

Anomaly detection vs Supervised learning:

Anomaly detection	Supervised learning
→ Very small number of positive examples, → Large number of negative examples.	Large no. of positive & negative examples.
→ Many different types of anomalies	Sufficient positive examples for learning & prediction

Choosing features for Anomaly detection:

Non-Gaussian features → Transform them to see if we get Gaussian
($x \to \log(x)$ for example)

Error analysis:

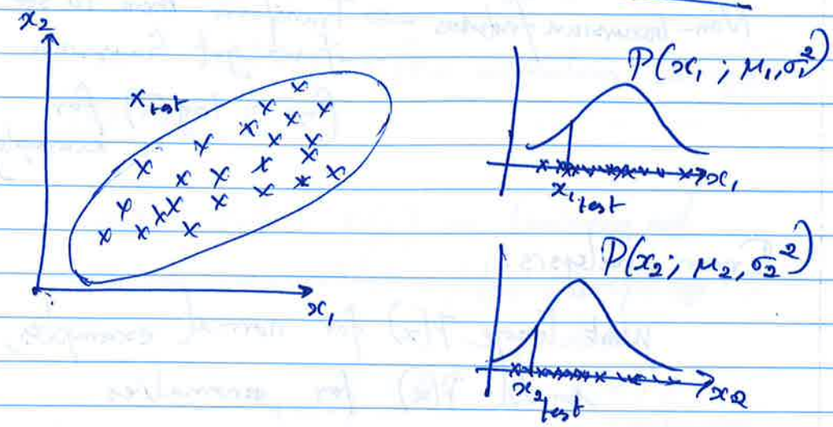
Want large $p(x)$ for normal examples,
small $p(x)$ for anomalies.

Problem: $p(x)$ is comparable for both

New features:

$$x_j = \frac{\text{CPU load}}{\text{Network traffic}}$$

Multivariate Gaussian Distribution:



Problem:
Individual features by themselves are not anomalous, but their combination is

$\downarrow$

$\prod_{j=1}^{n} P(x_j; \mu_j, \sigma_j^2)$ fails to catch this

(each individual $P(x_j)$ will be high enough).

Solution: Multivariate Gaussian distribution

$x \in \mathbb{R}^n$. Don't model $P(x_1), P(x_2), \ldots, P(x_n)$ separately.

Model $P(x)$ all in one go.

Params: $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$ $\hookrightarrow$ Covariance Matrix

$$P(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

$|\Sigma| \rightarrow$ Determinant of Σ

$\Sigma \rightarrow$ Covariance matrix $(n \times n)$

1) Parameter fitting:

Training set: $\{x^{(1)}, \ldots, x^{(m)}\}$

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

$\downarrow$ Covariance matrix

2) New example x:

$$p(x) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

$p(x) < \epsilon \Rightarrow$ anomaly

Original model (Univariate Gaussian)

$$P(x) = \prod_{j=1}^{n} P(x_j ; \mu_j, \sigma_j^2)$$

↓

This corresponds to multivariate Gaussian $P(x; \mu, \Sigma)$

when $\Sigma = \begin{bmatrix} \sigma_1^2 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & & \sigma_n^2 \end{bmatrix}$

↓

Covariance matrix

↓

Meaning, features are linearly independent

Univariate Gaussian	Multivariate Gaussian
- Manually create features to capture relationship between features	Automatically captures correlation b/w features.
- Computationally cheaper	Computationally expensive (computing Σ^{-1})
- Okay if m (training set size) is small.	- $m > n$, otherwise Σ is non-invertible

Recommender Systems:

→ Content-based recommendation:

→ Create features for each item to predict

→ Create feature vector

→ For each user, learn params $\theta^{(j)} \in \mathbb{R}^3$

→ Prediction for user j:

$$(\theta^{(j)})^T \cdot x$$

$r(i,j) = 1$ if user j has rated movie i

$y^{(i,j)}$ = rating by user j on movie i

$\theta^{(j)}$ = parameter vector for user j

$x^{(i)}$ = feature vector for movie i

For user j, movie i, predicted rating:

$$(\theta^{(j)})^T (x^{(i)})$$

$m^{(j)}$ = no. of movies rated by user j

To learn $\theta^{(j)}$:

$$\min_{\theta^{(j)}} \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T (x^{(i)}) - y^{(i,j)} \right)^2 + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

(Linear regression)

To learn $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$:

$$\min_{\theta^{(1)},\dots,\theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(\theta_k^{(j)}\right)^2$$

$$J\left(\theta^{(1)}, \dots, \theta^{(n_u)}\right)$$

Gradient descent:

$$\theta_k^{(j)} = \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right) x_k^{(i)} \quad (\text{for } k=0)$$

$$\theta_k^{(j)} = \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right) x_k^{(i)} + \lambda \theta_k^{(j)} \right) \quad (\text{for } k \neq 0)$$

Collaborative Filtering: → Learn features for item x

Given $\theta^{(1)}, \dots, \theta^{(n_u)}$, to learn $x^{(i)}$:

$$\min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2$$

Given $\theta^{(1)}, \dots, \theta^{(n_u)}$, to learn $x^{(1)}, \dots, x^{(n_m)}$:

$$\min_{x^{(1)},\dots,x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2$$

Collaborative Filtering Algorithm:

1. Given $x^{(1)}, \dots, x^{(n_m)}$, estimate $\theta^{(1)}, \dots, \theta^{(n_u)}$
2. Given $\theta^{(1)}, \dots, \theta^{(n_u)}$, estimate $x^{(1)}, \dots, x^{(n_m)}$
3. Repeat (1) & (2) till convergence.

(Or)

Minimizing $x^{(1)}, \dots, x^{(n_m)}$ and $\theta^{(1)}, \dots, \theta^{(n_u)}$ simultaneously

$$J\left(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}\right) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(\theta_k^{(j)}\right)^2$$

$$\min_{\substack{x^{(1)},\dots,x^{(n_u)} \\ \theta^{(1)},\dots,\theta^{(n_u)}}} J\left(x^{(1)}, \dots, x^{(n_u)}, \theta^{(1)}, \dots, \theta^{(n_u)}\right)$$

Collaborative filtering algo:

1.) Initialize $x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}$ to small random values.

2.) Minimize $J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$ using gradient descent (or other optimization).

3.) For a user with params θ and a movie with learned features x, predict rating as $\theta^T x$

Vectorization: Low Rank Matrix Factorization:

$$Y = \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}$$

such that $y^{(i,j)}$ = rating of movie i by user j

Predicted ratings:

	User 1	User 2	...	User n_u
Movie 1	$(\theta^{(1)})^T(x^{(1)})$	$(\theta^{(2)})^T(x^{(1)})$	...	$(\theta^{(n_u)})^T(x^{(1)})$
Movie 2	$(\theta^{(1)})^T(x^{(2)})$	$(\theta^{(2)})^T(x^{(2)})$	...	$(\theta^{(n_u)})^T(x^{(2)})$
:	:			
Movie n_m	$(\theta^{(1)})^T(x^{(n_m)})$	$(\theta^{(2)})^T(x^{(n_m)})$	...	$(\theta^{(n_u)})^T(x^{(n_m)})$

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(n_m)})^T — \end{bmatrix}, \quad \Theta = \begin{bmatrix} — (\theta^{(1)})^T — \\ — (\theta^{(2)})^T — \\ \vdots \\ — (\theta^{(n_u)})^T — \end{bmatrix}$$

Predicted ratings $= X\Theta^T$

↓

Low-rank matrix factorization.

Mean Normalization:

→ If a user hasn't rated any movie, $y^{(i,j)} = ?$ for all movies i.

↳ $(\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2$ term of cost fn., is zero

‼ Not useful.

$Y \rightarrow$ mean normalize the ratings for each movie.

$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_{n_m} \end{bmatrix}$, $\mu_i \rightarrow$ mean rating of movie i

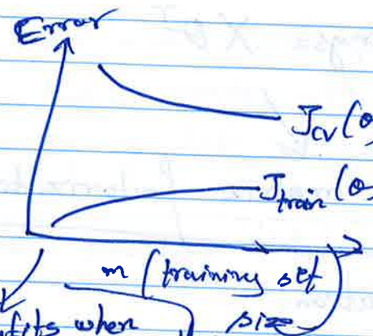
For user j on movie i, predict $(\theta^{(j)})^T (x^{(i)}) + \mu_i$

Week 10

Learning with large datasets:

→ First check if ~~large data~~ more training examples will help with learning

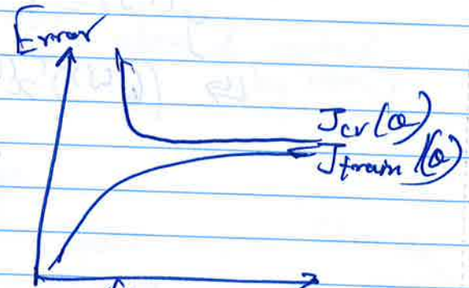
→ Plot the learning curve & verify



Overfits when (Variance) m is small

↓↓

More data will help



Underfits even with small m (High bias)

⇓

Increasing m won't help

Stochastic Gradient Descent (SGD):

Linear regression gradient descent:

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j = \theta^T x$$

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)}$$

↓

for (j=0, .. n)

}

For large m, computing $\frac{\partial}{\partial \theta_j} J(\theta)$ is expensive.

↳ Batch gradient descent

SGD:

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$J_{train}(\theta) = \frac{1}{m} \sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$$

1. Randomly shuffle dataset.
2. Repeat {

for i = 1, .., m {

$$\theta_j := \theta_j - \alpha \underbrace{\left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)}}_{\frac{\partial}{\partial \theta_j} cost(\theta, (x^{(i)}, y^{(i)}))}$$

(for j = 0, .., n)

}}

Mini-batch Gradient Descent:

Batch Gradient Descent → Use all m examples for each iteration

Stochastic Gradient Descent → Use 1 example in each iteration

Mini batch Gradient Descent → Use b examples in each iteration
(b = mini batch size)

Say $b = 10$, $m = 1000$

Repeat {

for $i = 1, 11, 21, 31, \ldots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} \left(h_\theta(x^{(k)}) - y^{(k)} \right) \cdot x_j^{(k)}$$

(for every $j = 0, \ldots, n$)

}

}

→ Better vectorization over SGD.

→ But, need to fit another param 'b'.
(mini batch size)

SGD convergence check:

Batch gradient descent convergence check:

~~→ Plot $J_{train}(\theta)$ as a function~~

→ Plot $J_{train}(\theta)$ vs no. of iterations

→ will always decrease (non-increasing) with each iteration.

~~Note~~

SGD → $J(\theta)$ can increase after each iteration.

↳ Checking for convergence:

→ $\text{cost}\left(\theta, (x^{(i)}, y^{(i)})\right)$
$= \frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

→ During learning, compute $\text{cost}\left(\theta, (x^{(i)}, y^{(i)})\right)$ before updating θ using $(x^{(i)}, y^{(i)})$.

→ Every 1000 iterations, plot $\text{cost}\left(\theta, (x^{(i)}, y^{(i)})\right)$ averaged over the last 1000 examples.

Learning rate in SGD:

→ Slowly decrease α (learning rate) over time to make θ converge better.

$$\left(\text{eg.: } \alpha = \frac{\text{const 1}}{(\text{iteration Number} + \text{const 2})} \right)$$

Online Learning:

→ One example at a time

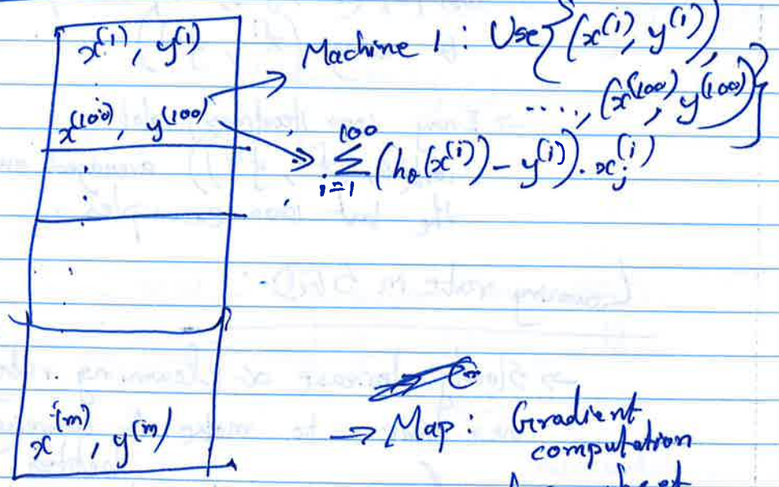
→ Use it to update params, and then discard the example (just one iteration or update, example never used again)

→ Adapt w

→ No fixed training set.

→ Can adapt to changes in model over time.

Map reduce & Data parallelism:



→ Map: Gradient computation for subset of examples.

→ Reduce: Average all gradients, update params.

Week 11

Photo OCR:

Pipeline:

Step 1: Text detection

Step 2: Character segmentation

Step 3: Character classification

Step 4: (Optional) Spell correction

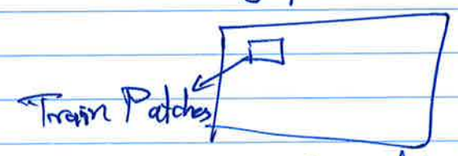
Image → Text detection → Character Segmentation → Character Recognition

Sliding Windows:

Text detection:

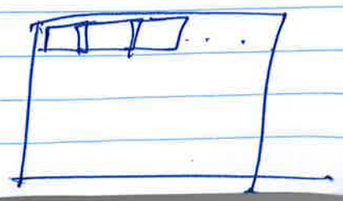
Train:

→ Image patches with and without text



→ Sliding window detection for text

- Sliding window over the entire image to get the patch
- Stride



→ 1D sliding window for character segmentation.

→ Classify characters

Artificial Data Synthesis:

Ceiling Analysis: